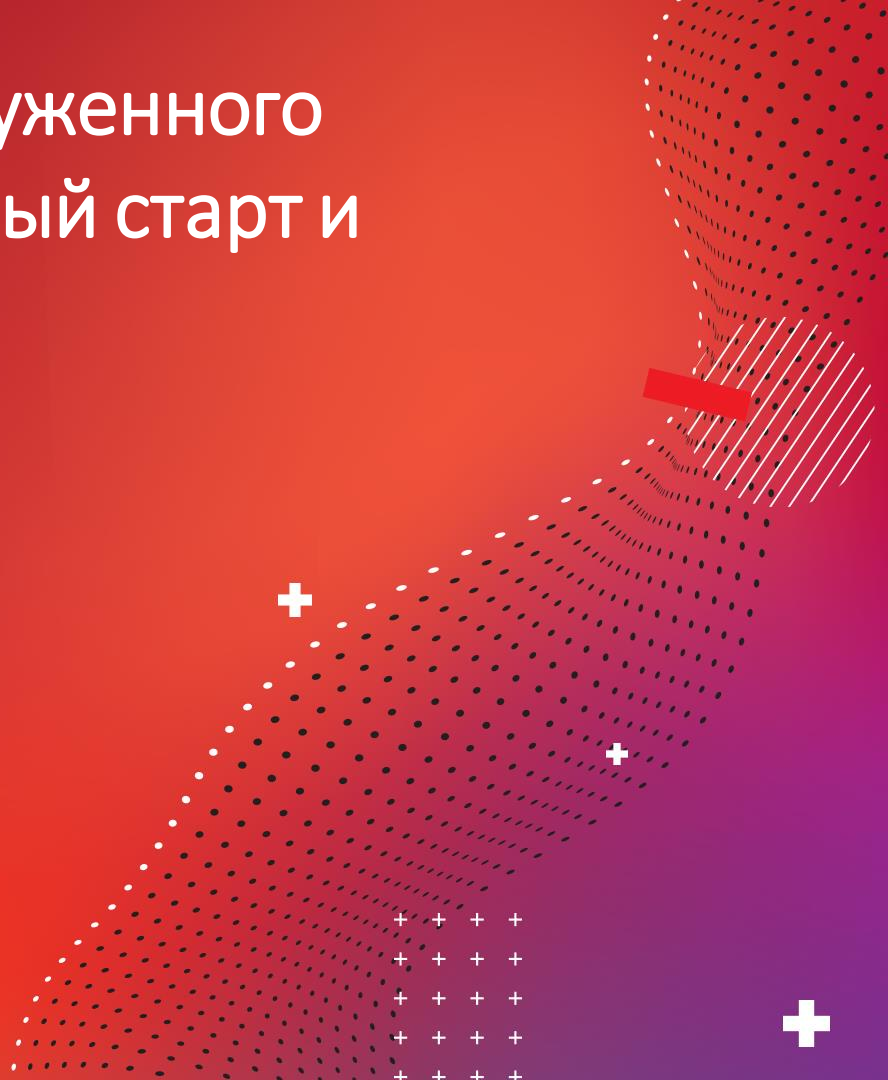


Язык Rust для высоконагруженного сетевого сервиса — быстрый старт и стремительный полет

Александр Сербул



HighLoad++
Весна 2021



О чем поговорим

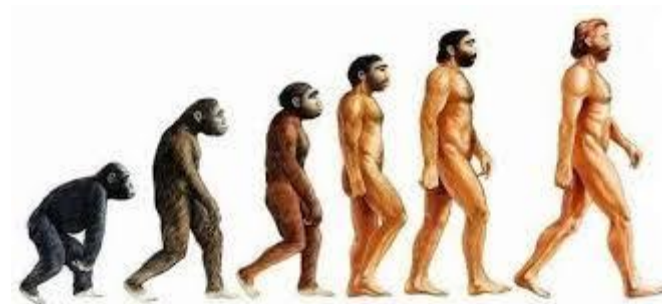
- Что происходит в IT и языках программирования «на самом деле»
- История проекта отправки push-уведомлений
- Как мы развивали архитектуру проекта: от bash-forks до Rust/Tokio
- Одновременно поговорим о высоконагруженной работе с сетью в php, java, python, nodejs, go lang и почему мы взяли Rust
- Как быстро понять Rust и его сравнение с другими ЯП
- Как начать решать задачи на Rust как можно быстрее, Cargo
- Подводные камни в Rust и способы их преодоления
- Выводы, рекомендации для будущих проектов и нагруженных сетевых архитектур



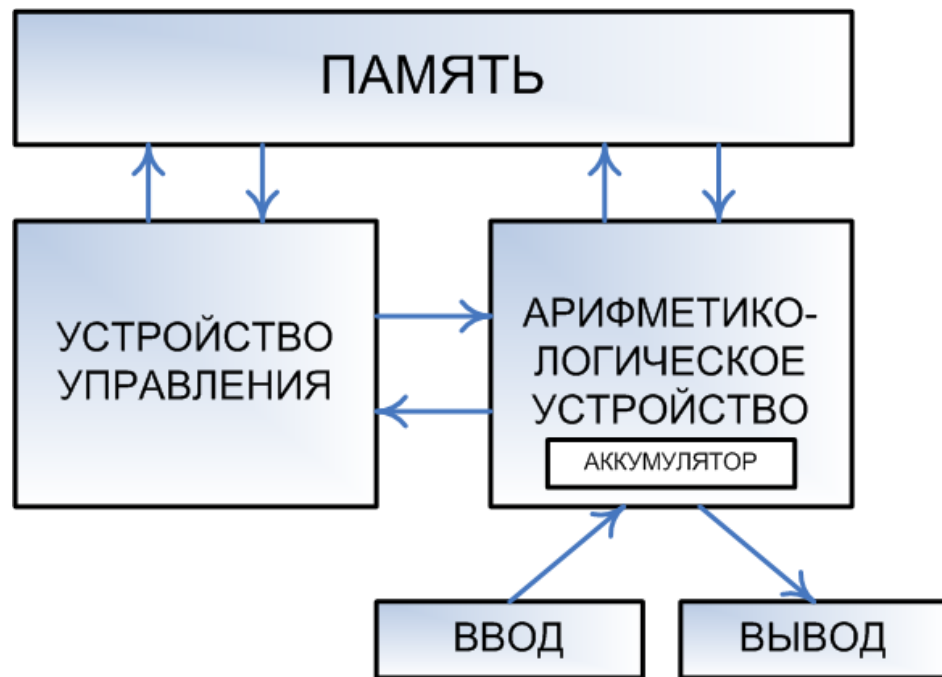
**Что происходит
в IT и языках
программирования
«на самом деле» —
и что подарил миру Rust?**

Как все начиналось...

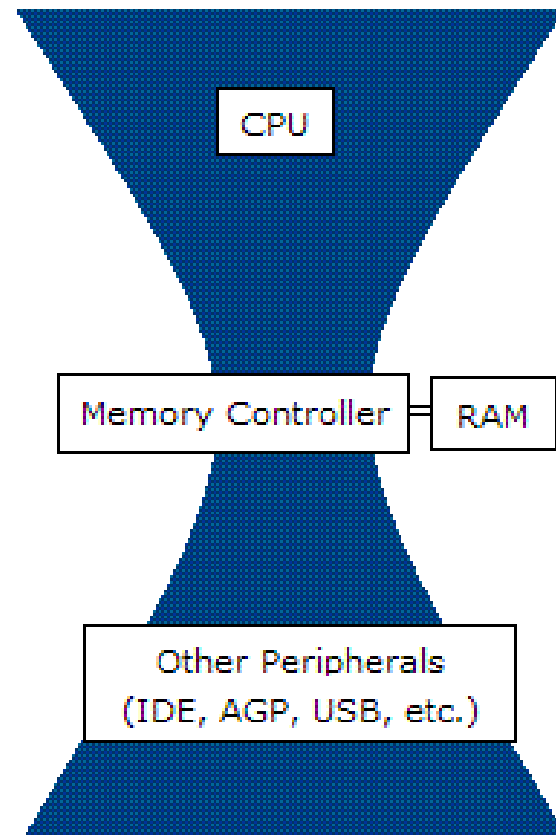
- Архитектура фон Неймана и машина Тьюринга
- Процедурные языки программирования
- Объектно-ориентированные языки программирования
- Функциональные языки программирования
- Гибридные языки программирования



Все не заладилось с самого начала...



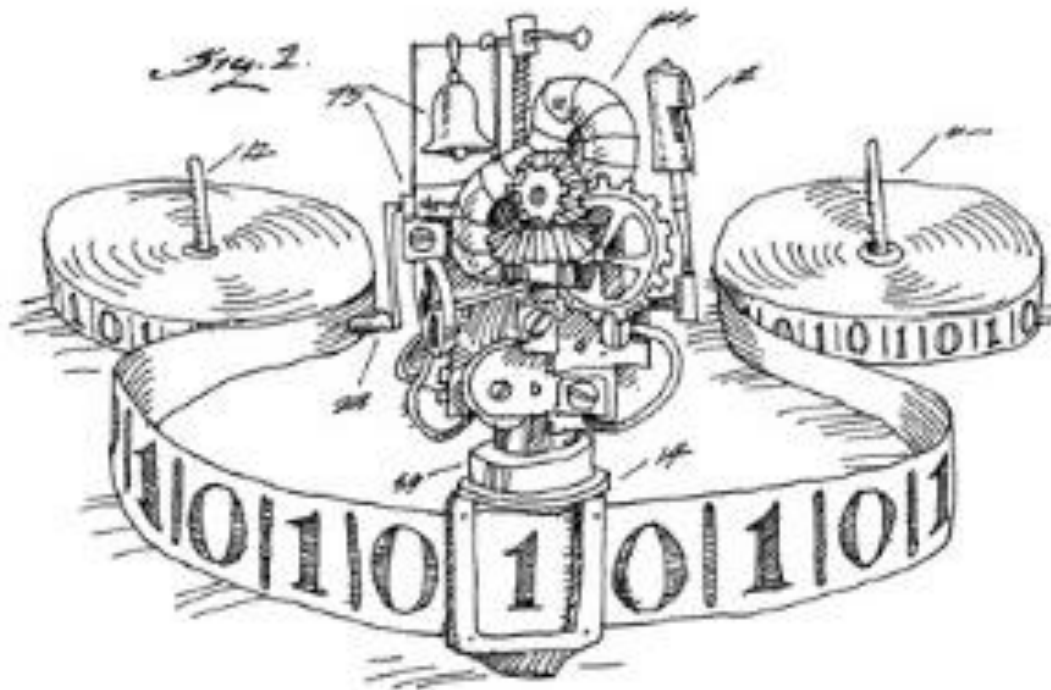
Архитектура фон Неймана





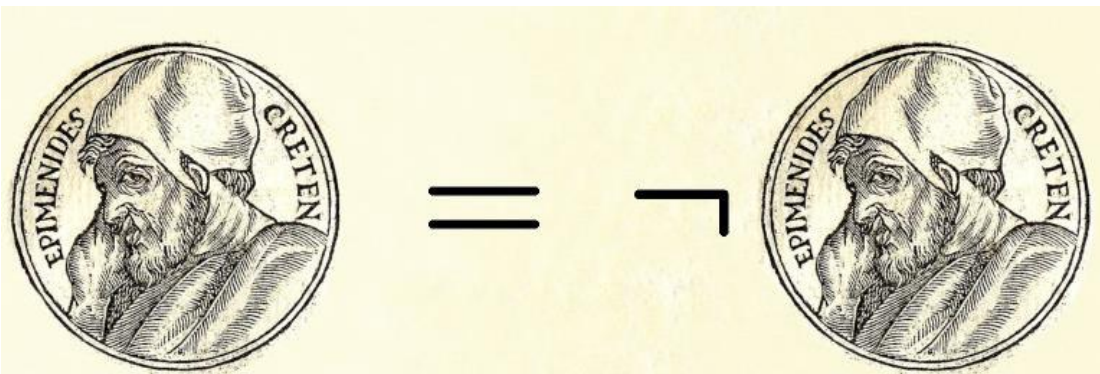
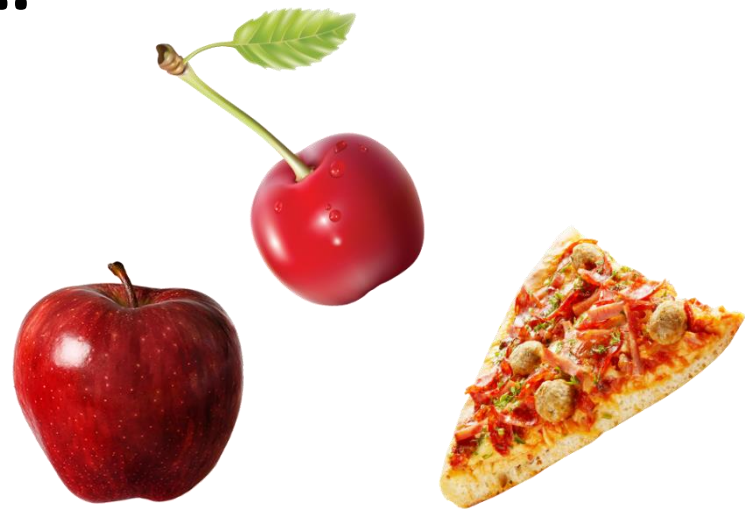
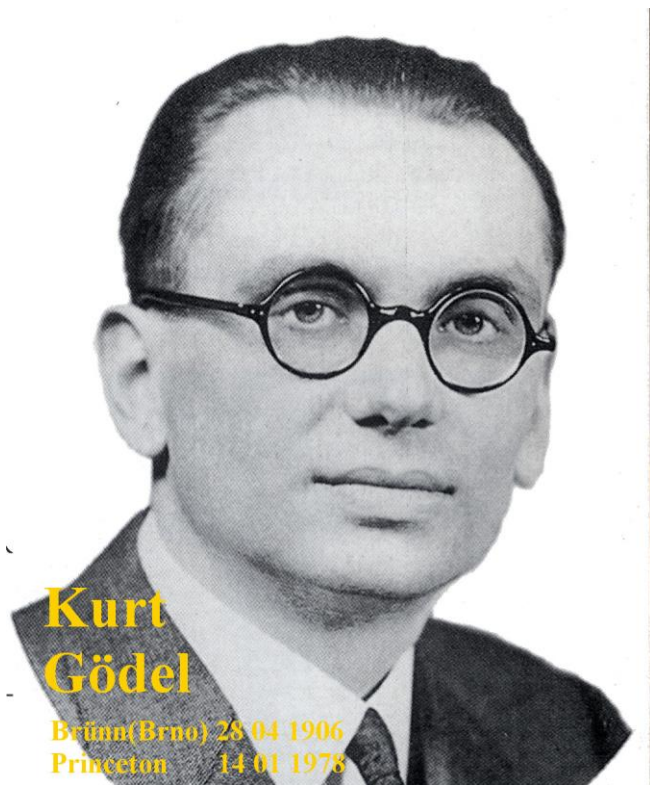
Машина Тьюринга

Универсальный вычислитель...



Машина Тьюринга

Теорема Гёделя о неполноте...



Современный код





современный Код

Управление сложностью

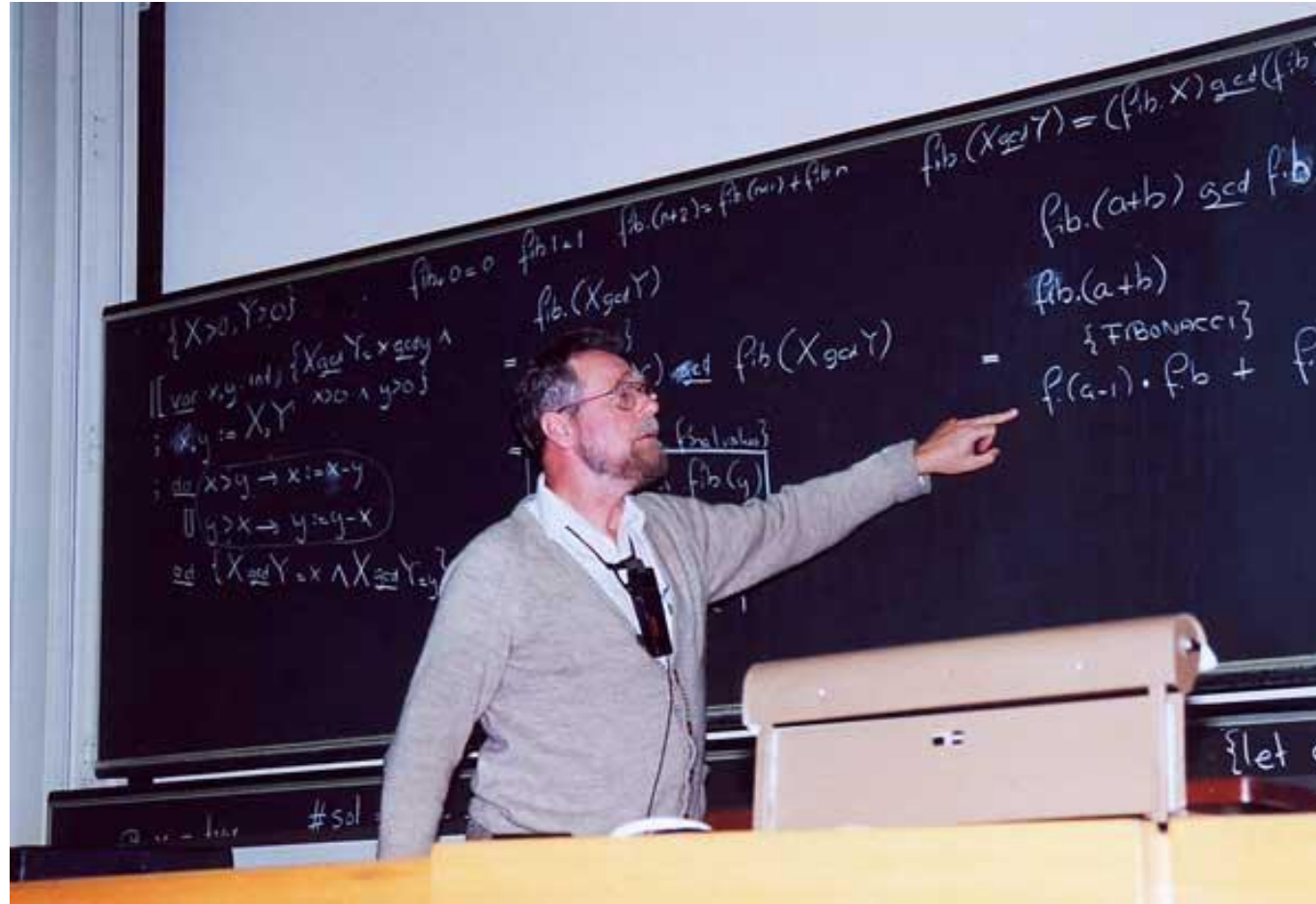
- инкапсуляция (все лишнее – скрыто поглубже), но Python, JavaScript опровергают пользу этого
- абстракция (сложное становится простым, фасады), но появляются текущие абстракции
- иерархии абстракций, но многие задачи становятся хуже от иерархий классов
- нужно большое количество автотестов, но как ими покрыть все и что происходит на практике

Haskell: “Quicksort”

```
quicksort [] = []  
quicksort (x:xs) = quicksort lesser ++ [x] ++ quicksort greater  
  where  
    lesser = [ y | y <- xs, y < x ]  
    greater = [ y | y <- xs, y >= x ]
```

А могло же быть так: просто и сексуально

Эдсгер Дейкстра



Чем все закончилось...

- Человечество – просиживает в соцсетях и в онлайн-играх
- Настоящее программирование – требует сильного напряжения ума и, желательно, знания математики
- Взрослые люди – не хотят учиться, а это нужно для успеха в IT
- Языки программирования – создаются методом копирования идей, борьба за разработчиков
- Думать и учить логически строгие ЯП типа Haskell – никто не хочет

Войны технологий – усугубляют сложность

- Вместо сотрудничества – конкуренция на рынке
- Вместо «развития» языков (Scala, Rust), отмечается «межвидовая мутация» (Node.js) и параллельное изобретение велосипедов (PHP, Ruby)
- Масштабные религиозные войны: Java vs C#, PHP vs Python vs Ruby
- Войны фреймворков: Angular vs React JS
- “Зачем делать просто, если можно сложно?”



Mathematicians stand on each others' shoulders and computer scientists stand on each others' toes.

— *Richard Hamming* —

AZ QUOTES

«Шовинизм» в программировании

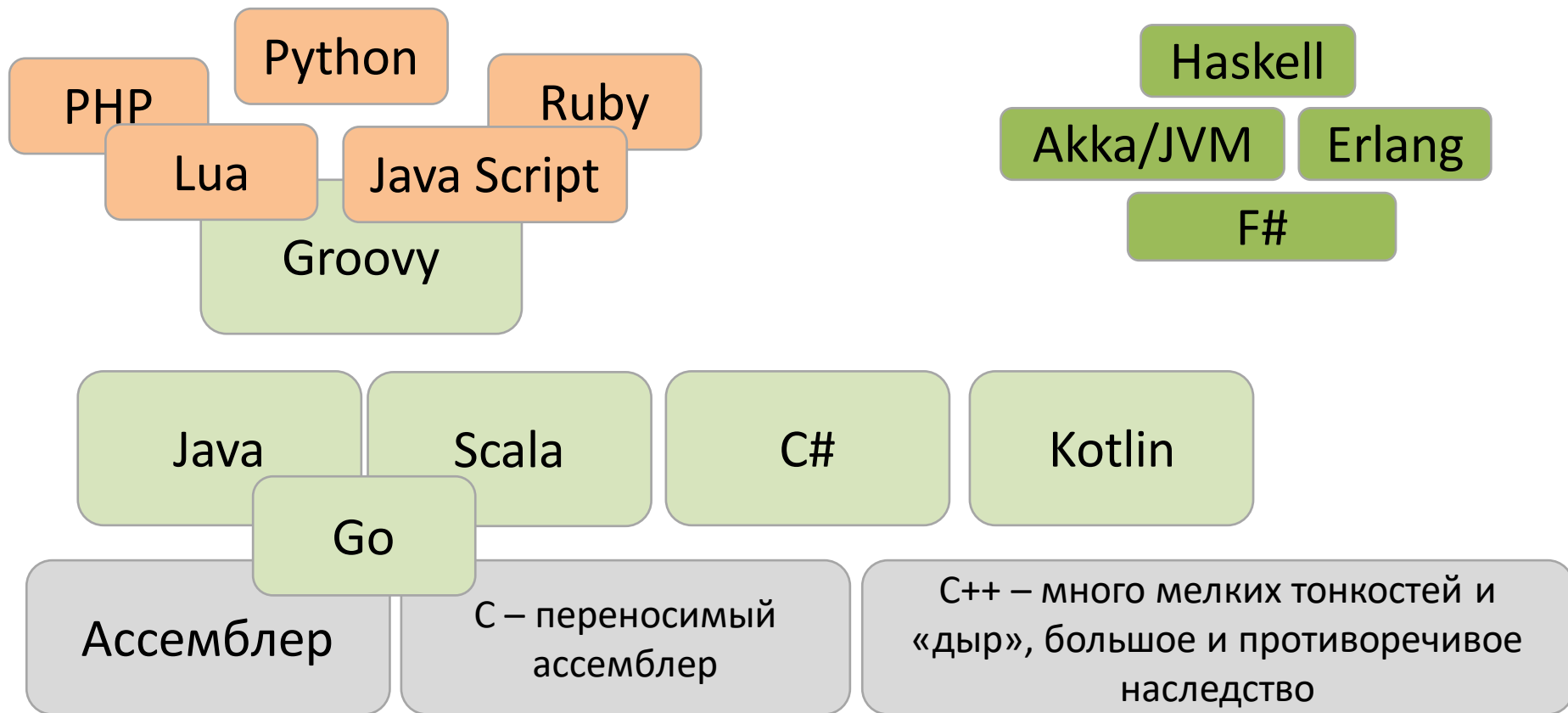
- Вы писали 5 лет на PHP и делали сайты на Битриксе... Но почему, за деньги что-ли? – на работу не беру
- Вы только байтики умеете на C гонять? – в нашу компанию вы не подходите
- Только Java/Spring, только своих... – и пофиг на скорость, качество, результат
- C#/Windows forever
- Гошечка, милая гошечка...

Докладчик сошел с ума 😊

Все свободны.

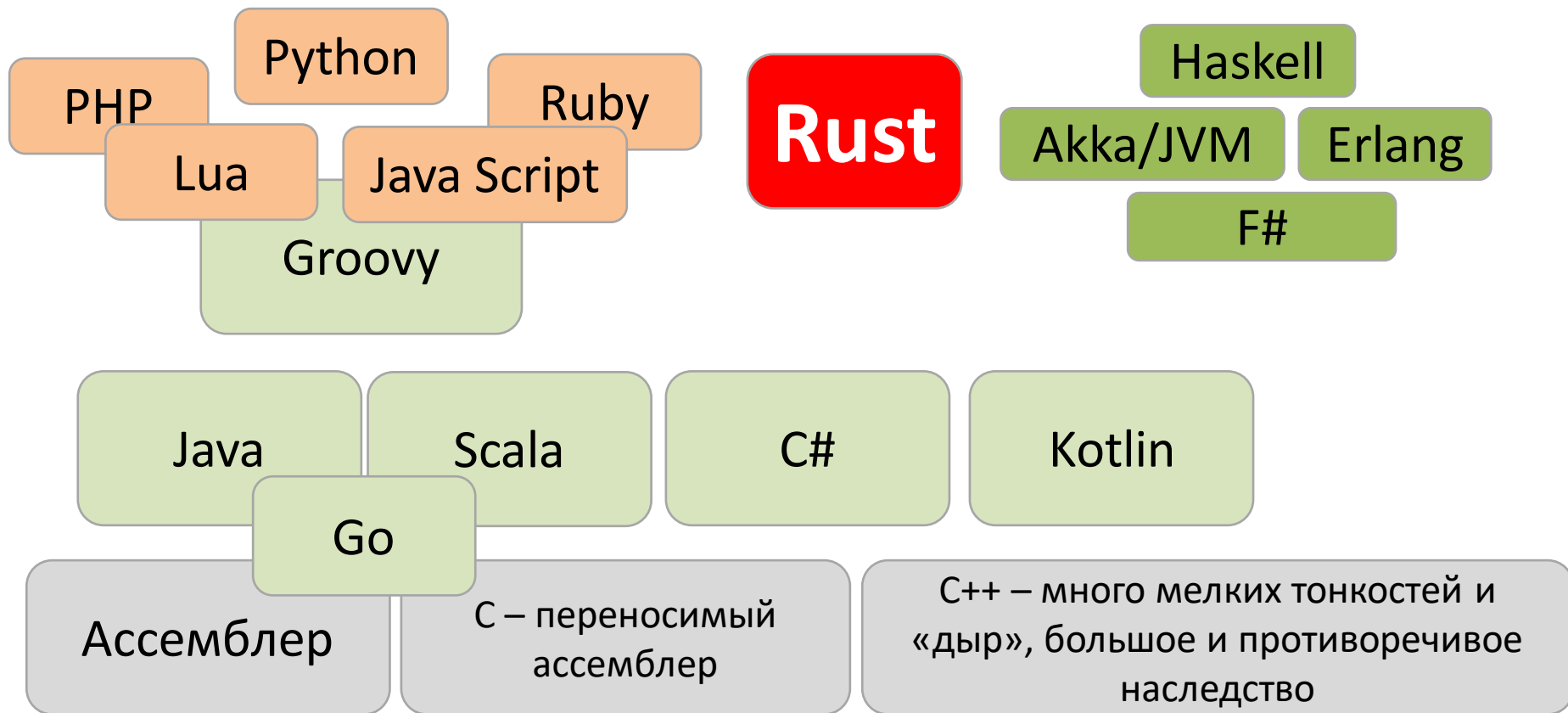
Спасибо за внимание!

Выразительность языков программирования





Выразительность языков программирования



Выразительность в Rust

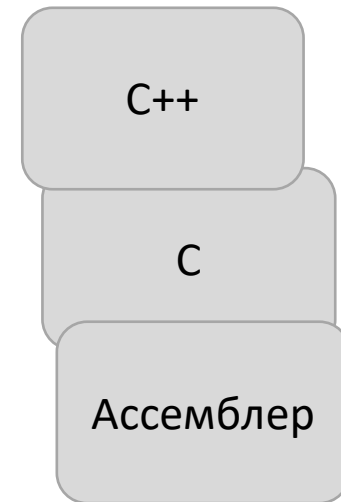
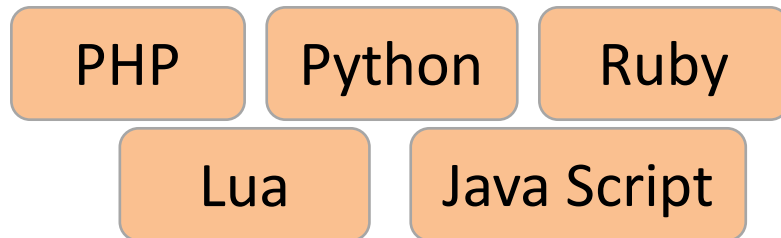
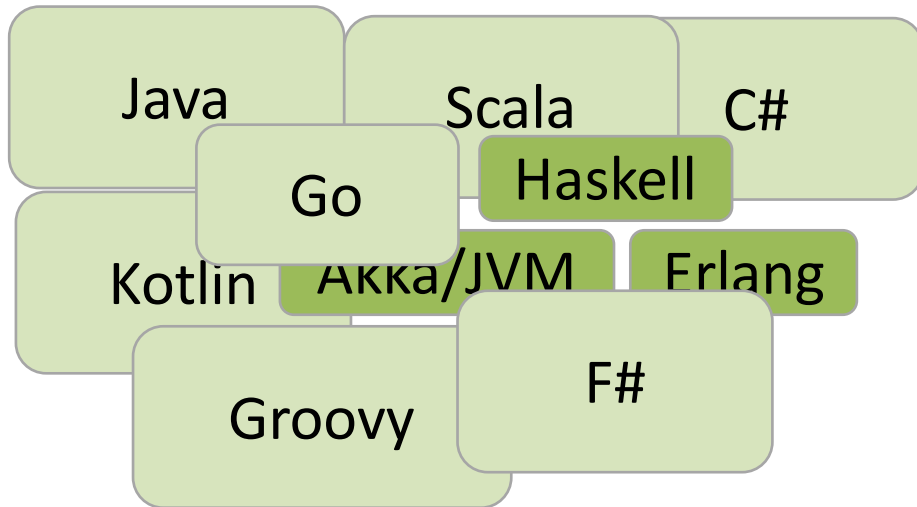
- Zero-cost (!) абстракции: итераторы, фильтры, мап-редьюсы... (одна из сильных сторон C++)
- Автовывод типов
- Нет NULL, даже как концепции
- Алгебраические типы данных (ADT)
- Надежный pattern-matching по ADT
- Защита от «опасных» операций с массивами, числами (переполнения...); “panic!”
- Компилятор дает надежные гарантии безопасности при работе с памятью и автоматически освобождает ее – нет сборщика мусора
- Очень мощный и строгий макро-язык
- Нельзя напороться на «**Undefined Behavior**» (кроме как в unsafe режиме, но это нужно прямо постараться)

MY LANGUAGE



YOUR BUGS

Управление памятью





Управление памятью



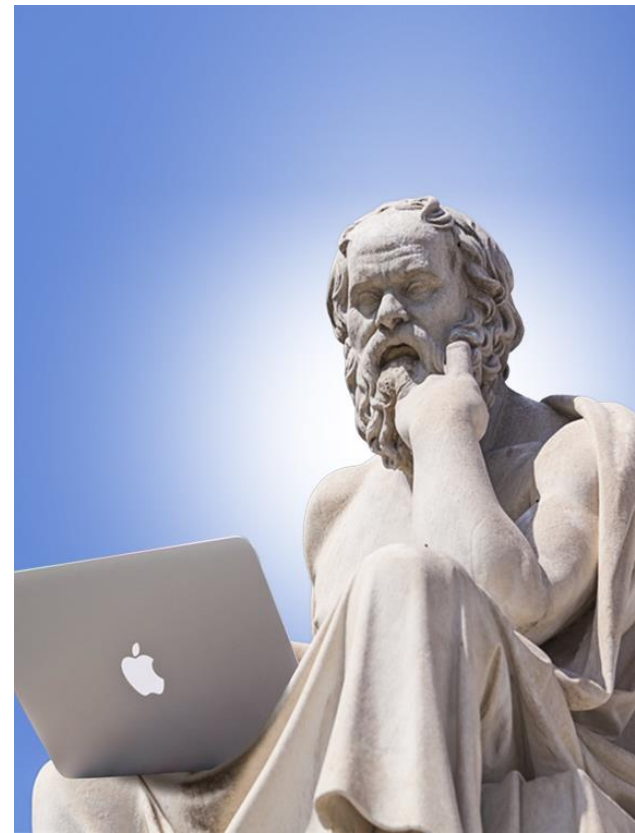
Управление памятью в Rust

- Владение объектами (ownership)
- Передача объектов между функциями (move) и автоматический вызов «деструкторов», когда «пора»
- Заимствование объектов (borrowing) при передаче по ссылке, в функции, ссылочная целостность
- Slices, lifetimes
- **No dangling references**

Как быстро понять Rust и его сравнение с другими ЯП

Философии

- Python
- JavaScript/Node.js
- TypeScript
- PHP
- Java/C#
- Scala/Kotlin
- C/C++
- Haskell/ML
- Rust



Как набить руку в Rust

- “Rust by example”: <https://doc.rust-lang.org/stable/rust-by-example/>
- “The Book”: <https://doc.rust-lang.org/book/>
- “The Rustonomicon”: <https://doc.rust-lang.org/nomicon/>

Первые 2 ссылки лучше пройти с руками в консоли 2-3 раза и потом все открывается и дальше легко.

Самое непривычное для понимания: аффинные типы данных, алгебраические типы данных. Полезен опыт изучения FP.

Последняя ссылка самая сложная и тяжелая, но ее можно ... не читать 😊


```
// `a` is a pointer to a _heap_ allocated integer  
let a = Box::new(5i32);
```

```
println!("a contains: {}", a);
```

```
// *Move* `a` into `b`
```

```
let b = a;
```

```
// The pointer address of `a` is copied (not the data) into `b`.
```

```
// Both are now pointers to the same heap allocated data, but
```

```
// `b` now owns it.
```

```
// Error! `a` can no longer access the data, because it no longer owns the
```

```
// heap memory
```

```
//println!("a contains: {}", a);
```

```
// _Stack_ allocated integer  
let x = 5u32;  
  
// *Copy* `x` into `y` - no resources are moved  
let y = x;  
  
// Both values can be independently used  
println!("x is {}, and y is {}", x, y);
```

Заимствование

```
// This function borrows an i32
fn borrow_i32(borrowed_i32: &i32) {
    println!("This int is: {}", borrowed_i32);
}

fn main() {
    // Create a boxed i32, and a stacked i32
    let boxed_i32 = Box::new(5_i32);
    let stacked_i32 = 6_i32;

    // Borrow the contents of the box. Ownership is not taken,
    // so the contents can be borrowed again.
    borrow_i32(&boxed_i32);
    borrow_i32(&stacked_i32);

    {
        // Take a reference to the data contained inside the box
        let _ref_to_i32: &i32 = &boxed_i32;
    }
}
```

Аффинные типы данных – проникли гораздо глубже

- Концепции: ownership, move, borrowing – активно применяются в многопоточном программировании и в других местах (итераторы и др.)
- Забыть освободить Mutex – невозможно, не скомпилируется 😊
- Поработать с инвалидированным итератором – невозможно, не скомпилируется 😊
- Появилось больше времени на решение бизнес-задач

Как не поддаться унынию при изучении Rust

- Компилятор – дает гарантии на безопасное управление памятью, нужно этим «эффективно пользоваться»
- Многие вещи в начале будут непонятны – но если код скомпилировался, то будет работать ПРАВИЛЬНО
- Важно научиться «бороться с компилятором» и побеждать его
- Знание мелких деталей приходит с опытом, но важно понять ИДЕЮ ЯЗЫКА, ЕГО ПОСЫЛ

- развлечения с lifetimes и lifetime elision

```
// implicit
fn foo(x: &i32) {
}

// explicit
fn bar<'a>(x: &'a i32) {
}
```

```

struct Foo<'a> {
    x: &'a i32,
}

fn main() {
    let x; // -+ x goes into scope
           // |
    {     // |
        let y = &5; // ----+ y goes into scope
        let f = Foo { x: y }; // ----+ f goes into scope
        x = &f.x; // | | error here
    }           // ----+ f and y go out of scope
           // |
    println!("{}", x); // |
}                   // -+ x goes out of scope

```


Что я плохо понимаю до сих пор и это нестрашно

- «сложные» lifetimes

```
fn x_or_y<'a, 'b>(x: &'a str, y: &'b str) -> &'a str {
```

```
struct Foo<'a> {  
    x: &'a i32,  
}  
  
fn main() {  
    let y = &5; // this is the same as `let _y = 5; let y = &_y;`  
    let f = Foo { x: y };  
  
    println!("{}", f.x);  
}
```

Начало работы с Rust

- «Непробиваемый» компилятор
- Rust-plugin IntelliJ IDEA

C:\Users\alexs\Downloads\rustup-init.exe

```
Welcome to Rust!

This will download and install the official compiler for the Rust
programming language, and its package manager, Cargo.

Rustup metadata and toolchains will be installed into the Rustup
home directory, located at:

  C:\Users\alexs\.rustup

This can be modified with the RUSTUP_HOME environment variable.

The Cargo home directory located at:

  C:\Users\alexs\.cargo

This can be modified with the CARGO_HOME environment variable.

The cargo, rustc, rustup and other commands will be added to
Cargo's bin directory, located at:

  C:\Users\alexs\.cargo\bin

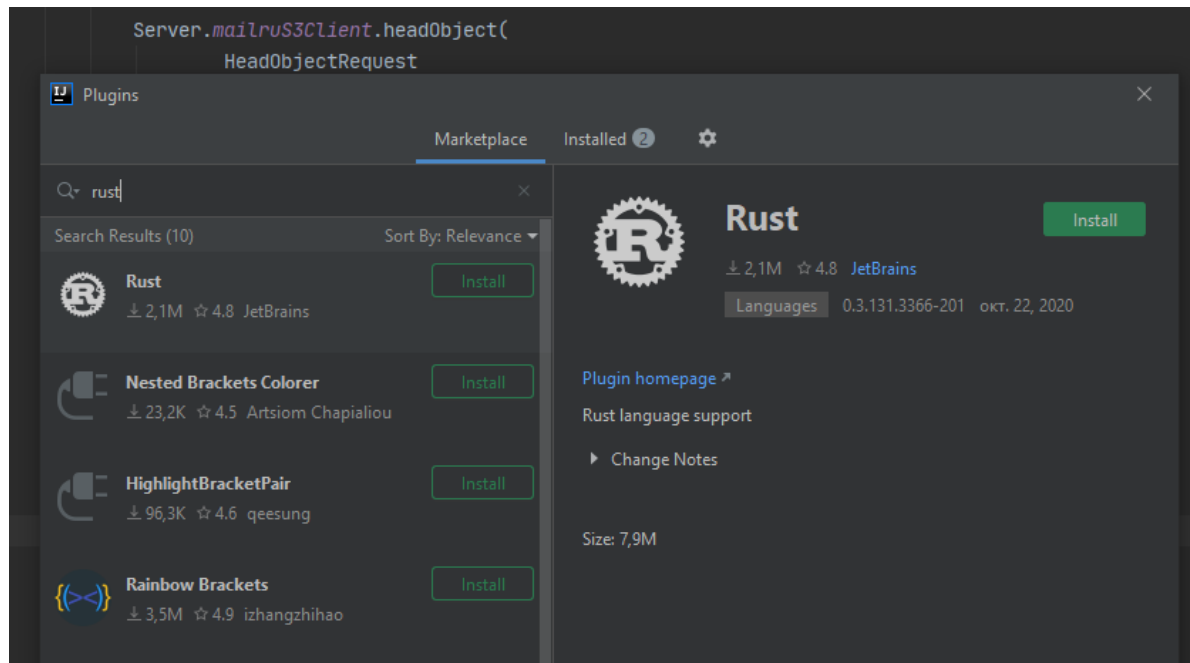
This path will then be added to your PATH environment variable by
modifying the HKEY_CURRENT_USER/Environment/PATH registry key.

You can uninstall at any time with rustup self uninstall and
these changes will be reverted.

Current installation options:

  default host triple: x86_64-pc-windows-msvc
  default toolchain:  stable (default)
                    profile: default
  modify PATH variable: yes




1) Proceed with installation (default)
2) Customize installation
3) Cancel installation
```



Сборка бинарника

```
1
2
3 fn main() {
4
5     println!("Hello, Highload!");
6
7 }
8
```

> Windows (C:) > Пользователи > alexs > rust_projects > hw

Имя	Дата изменения	Тип	Размер
 hw.exe	22.10.2020 12:41	Приложение	149 КБ
 hw.pdb	22.10.2020 12:41	Program Debug D...	1 084 КБ
 hw.rs	22.10.2020 12:38	Файл "RS"	1 КБ

```
C:\Users\alexs\rust_projects\hw>rustc hw.rs
```

```
C:\Users\alexs\rust_projects\hw>hw.exe
Hello, Highload!
```

Стандартная библиотека Rust



Crate std

Version 1.48.0 (7eac88abb
2020-11-16)

See all std's items

Re-exports

Primitive Types

Modules

Macros

Keywords

Crates

alloc


core

proc_macro

Modules

alloc	Memory allocation APIs
any	This module implements the <code>Any</code> trait, which enables dynamic typing of any <code>'static</code> type through runtime reflection.
array	Implementations of things like <code>Eq</code> for fixed-length arrays up to a certain length. Eventually, we should be able to generalize to all lengths.
ascii	Operations on ASCII strings and characters.
borrow	A module for working with borrowed data.
boxed	A pointer type for heap allocation.
cell	Shareable mutable containers.
char	A character type.
clone	The <code>Clone</code> trait for types that cannot be 'implicitly copied'.
cmp	Functionality for ordering and comparison.
collections	Collection types.
convert	Traits for conversions between types.
default	The <code>Default</code> trait for types which may have meaningful default values.
env	Inspection and manipulation of the process's environment.
error	Traits for working with Errors.
f32	This module provides constants which are specific to the implementation of the <code>f32</code> floating point data type.
f64	This module provides constants which are specific to the implementation of the <code>f64</code> floating point data type.
ffi	Utilities related to FFI bindings.
fmt	Utilities for formatting and printing <code>Strings</code> .
fs	Filesystem manipulation operations.
future	Asynchronous values.
hash	Generic hashing support.

The Rust community's crate registry

 [Install Cargo](#)

 [Getting Started](#)

Instantly publish your crates and install them. Use the API to interact and find out more information about available crates. Become a contributor and enhance the site with your work.

4 656 136 048

Downloads



50 015

Crates in stock



New Crates

ckb-lib-secp256k1
v0.1.0



executor-trait
v0.0.0



reactor-trait
v0.0.0



dokkoo
v0.1.2



Most Downloaded

rand



syn



libc



quote



Just Updated

ckb-capsule
v0.4.4



qp2p
v0.9.2



tibco_ems
v0.1.9



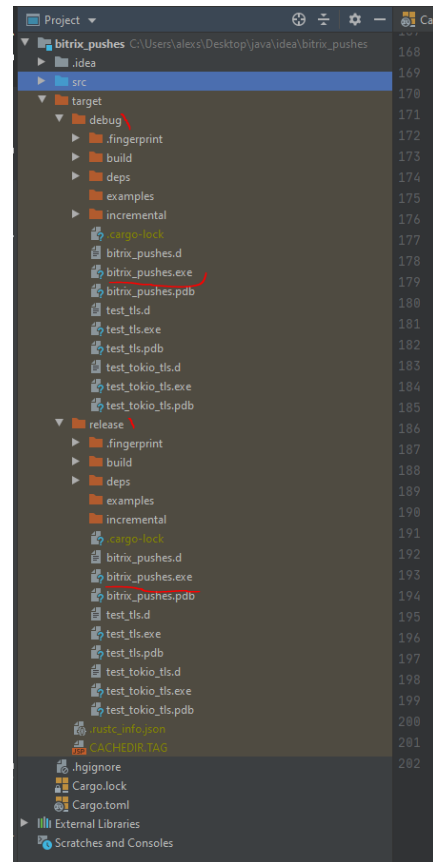
substrate-wasm-builder
v3.0.0



Менеджмент проекта, toolchain

- Cargo.toml
- Cargo.lock
- “cargo build --release”
- “rustup update”

```
[dependencies]
base64 = "0.10.1"
chrono = "0.4.9"
clap = "2.33.0"
futures = "0.1.29"
lazy_static = "1.4.0"
log = "0.4.8"
env_logger = "0.7.1"
regex = "1.3.1"
rusoto_sqs = "0.41.0"
rusoto_core = "0.41.0"
rustls = "0.16.0"
serde_json = "1.0.41"
signal-hook = "0.1.11"
tokio = "0.1.22"
tokio-core = "0.1.17"
tokio-rustls = "0.10.2"
webpki = "0.21.0"
webpki-roots = "0.18.0"
```



Автотестирование

- Unit tests
- Integration tests
- “cargo test”

```
1553     #[cfg(test)]
1554     mod tests {
1555
1556         use super::*;
1557
1558         #[test]
1559         fn test_push_type() {
1560
1561             assert_eq!(PushType::Unknown.value(), 0);
1562             assert_eq!(PushType::APNSSandbox.value(), 1);
1563             assert_eq!(PushType::APNSProd.value(), 2);
1564             assert_eq!(PushType::Google.value(), 3);
1565             assert_eq!(PushType::AppleV2Sandbox.value(), 4);
1566             assert_eq!(PushType::AppleV2Prod.value(), 5);
1567
1568             assert_eq!(PushType::by_num( n: 0), PushType::Unknown);
1569             assert_eq!(PushType::by_num( n: 1), PushType::APNSSandbox);
1570             assert_eq!(PushType::by_num( n: 2), PushType::APNSProd);
1571             assert_eq!(PushType::by_num( n: 3), PushType::Google);
1572             assert_eq!(PushType::by_num( n: 4), PushType::AppleV2Sandbox);
1573             assert_eq!(PushType::by_num( n: 5), PushType::AppleV2Prod);
1574             assert_eq!(PushType::by_num( n: 100), PushType::Unknown);
1575         }
1576     }
```

```
Finished test [unoptimized + debuginfo] target(s) in 2.44s
Running target\debug\deps\bitrix_pushes-4635fc3582b77339.exe

running 15 tests
test b24::amazon::tests::test_apci ... ok
test b24::amazon::tests::test_apple_response ... ok
test b24::amazon::tests::test_apiv2 ... ok
test b24::amazon::tests::test_apiv1 ... ok
test b24::amazon::tests::test_apci_p2 ... ok
test b24::amazon::tests::test_apci_p1 ... ok
test b24::amazon::tests::test_extract_raw_packets ... ok
test b24::amazon::tests::test_make_google_payload ... ok
test b24::amazon::tests::test_path2key_google ... ok
test b24::amazon::tests::test_push_type ... ok
test b24::amazon::tests::test_remove_spaces ... ok
test b24::amazon::tests::test_transform_raw_packets ... ok
test b24::amazon::tests::test_extract_transform ... ok
test b24::amazon::tests::test_transform_raw_packets2 ... ok
test b24::amazon::tests::test_check_tkey ... ok

test result: ok. 15 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out

Running target\debug\deps\test_tls-33786fbcca939adf.exe

running 0 tests

test result: ok. 0 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out

Running target\debug\deps\test_tokio_tls-b209aa9008c150fb.exe

running 0 tests

test result: ok. 0 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out
```

Unsafe Rust

The only things that are different in Unsafe Rust are that you can:

- Dereference raw pointers
- Call `unsafe` functions (including C functions, compiler intrinsics, and the raw allocator)
- Implement `unsafe` traits
- Mutate statics
- Access fields of `union`s

That's it. The reason these operations are relegated to Unsafe is that misusing any of these things will cause the ever dreaded Undefined Behavior. Invoking Undefined Behavior gives the compiler full rights to do arbitrarily bad things to your program. You definitely *should not* invoke Undefined Behavior.

И да, сторонние пакеты из Cargo могут содержать блоки «unsafe» и вызывать неопределенное поведение, но это случается на порядки реже, чем в связках python/c++ и т.п.

Итоги «плюшек» в Rust

- Пишешь быстро, как на python/php – автовывод типов
- Очень строгая типизация без NULLs с ADT/PM* – а не как структурная в Golang/TypeScript или слабая динамическая в PHP/JS или «строгая»-динамическая в Python
- Мощные гарантии компилятора: безопасная работа с памятью, авто-очистка памяти
- Нет неопределенного поведения и прочих сюрпризов – «безопасный» язык, как Python/PHP**
- Скорость как у C/C++!

* алгебраические типы данных/сопоставление по шаблону

** сюрпризы возможны в unsafe-режиме языка, но их на порядки меньше

Как мы развивали архитектуру проекта: от bash-forks до rust/tokio

Пуши – технологии



- **Apple Push Notification service (APNs)** – June 17, 2009; >= iOS 3.0, >= OS X v10.7 (Lion); max message – 256 B

<https://developer.apple.com/notifications/>

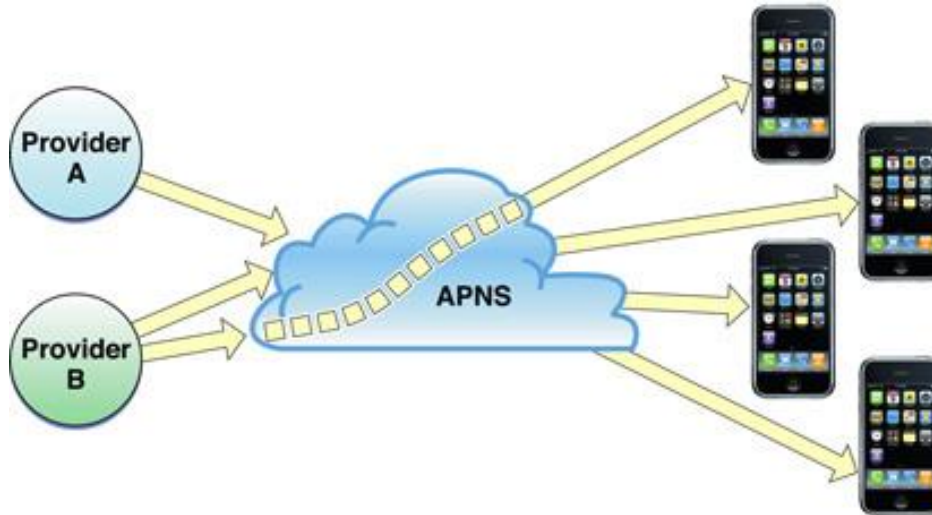
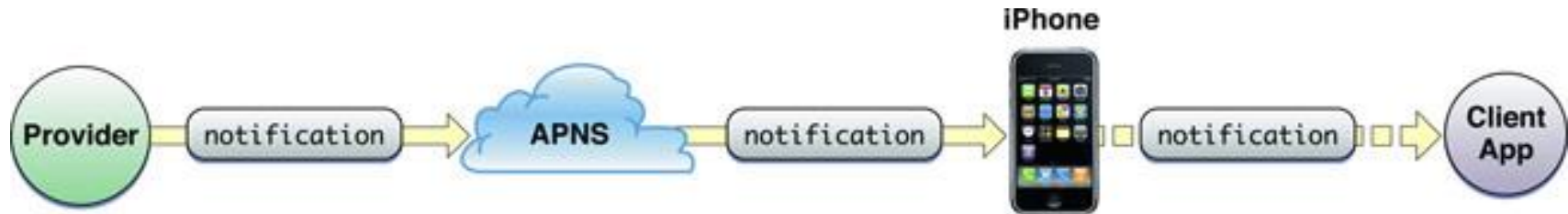


- **Google Cloud Messaging for Android (GCM)** – June 27, 2012; >= Android 2.2; max message – 4 KB

<http://developer.android.com/google/gcm/>

ранее – Android Cloud to Device Messaging (C2DM)

APNS



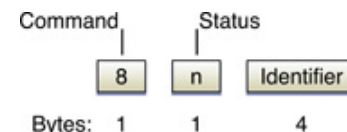
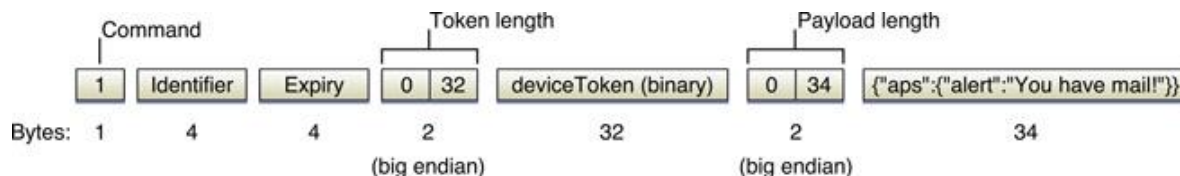
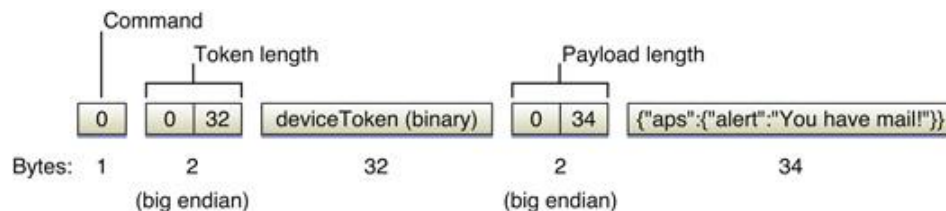
Идеи в основе архитектуры

1. Провайдер регистрируется в APNs, GCM и получает клиентский SSL/TLS-сертификат (ключ и т.п.)
 2. Провайдер обслуживает запросы многих мобильных приложений и идентифицирует себя в APNs, GCM с помощью сертификата
- Нам нужно поддерживать 10к – 100к и больше мобильных приложений



APNS – протокол

- Бинарный протокол
- 2 формата: простой и расширенный
- Пишем/читаем SSL/TLS-сокеты
- Возвращает инфу о невалидных токенах устройств



GCM – протокол

- JSON/plain text

POST -> `https://android.googleapis.com/gcm/send`

Запрос:

Content-Type:application/json

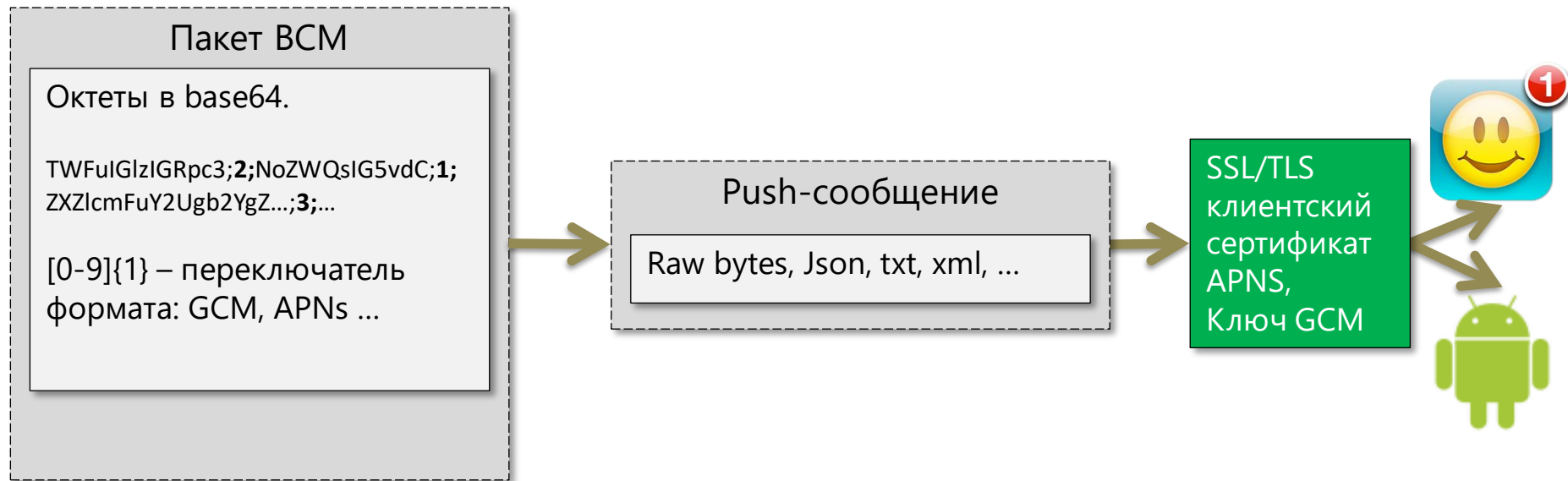
Authorization:key=*aSyB-luEai2WiUapxCs2Q0GZYZPu7Udno5aA

```
{
  "registration_ids" : ["APA91bHun4MxP5egoKMwt2KZFbAFUH-1RYqx..."],
  "data" : {
    ...
  },
}
```

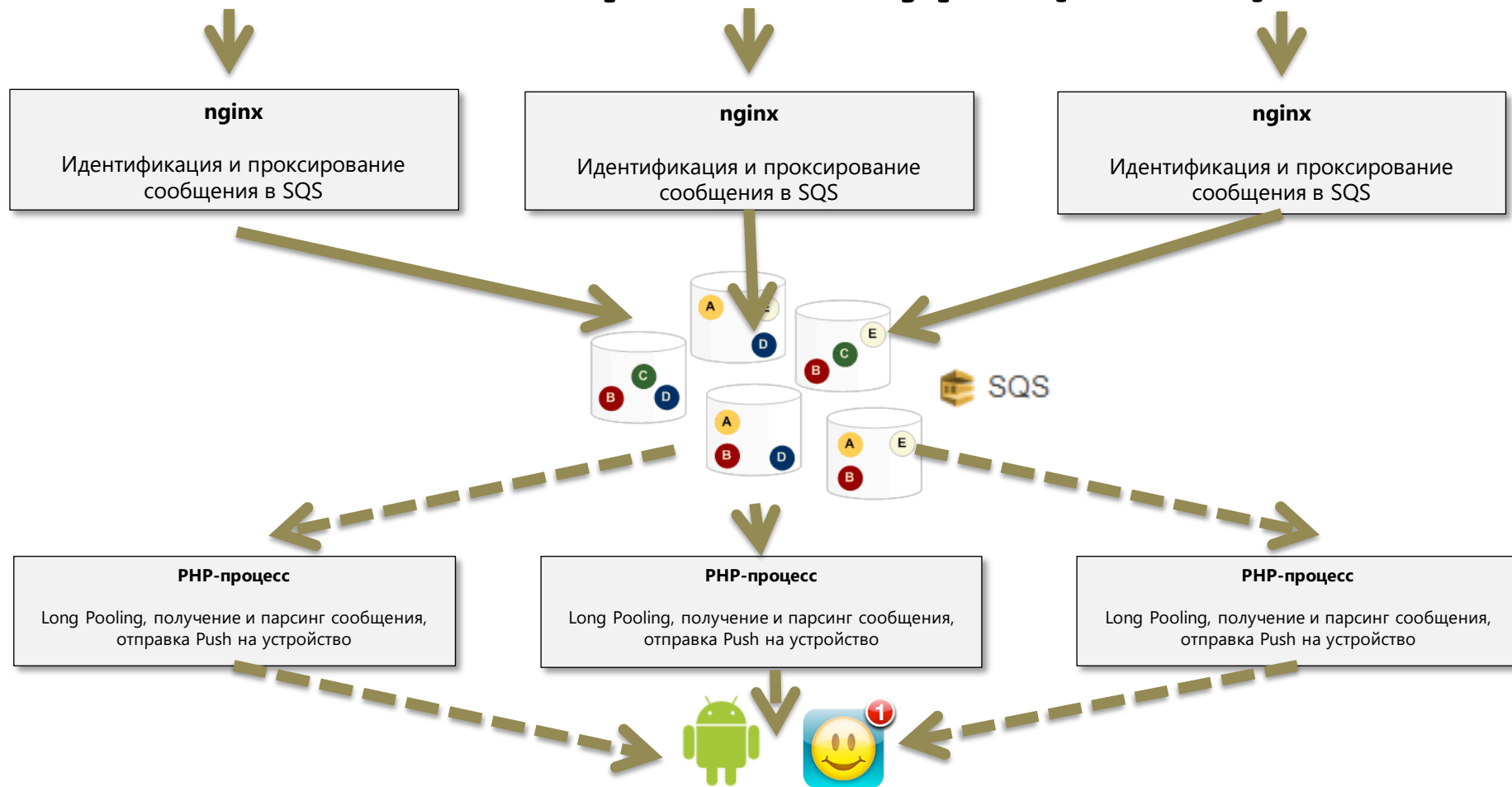
Ответ:

```
{ "multicast_id": 108,
  "success": 1,
  "failure": 0,
  "canonical_ids": 0,
  "results": [
    { "message_id": "1:08" }
  ]
}
```


Универсальный транспорт Битрикс24



Итоговая архитектура (2013)



Недостатки архитектуры

- Синхронная работа с сокетами при отправке push-уведомлений в GCM/APNs
 - За счет скриптовых языков – высокая нагрузка на CPU
- 1) Итерация 1: PHP форкает системный процесс отправки push
 - 2) Итерация 2: PHP начал работать с сокетами сам

Нагрузка:

до 1000 SQS-сообщений в секунду

до 1000-15 000 отправляемых пуш-уведомлений в секунду

Нагрузка, железо

Нагрузка:

до 1000 SQS-сообщений в секунду

до 1000-15 000 отправляемых пуш-уведомлений в секунду

- На PHP кластер жил за 6-8 серверах с 16-32 CPU и 16-32 ГБ ОЗУ и нагрузка постоянно росла
- На Rust – нагрузку пушей Битрикс24 держит **один** сервер 4 CPU 8 ГБ ОЗУ (процесс, он один, занимает 3.5 ГБ в ОЗУ)

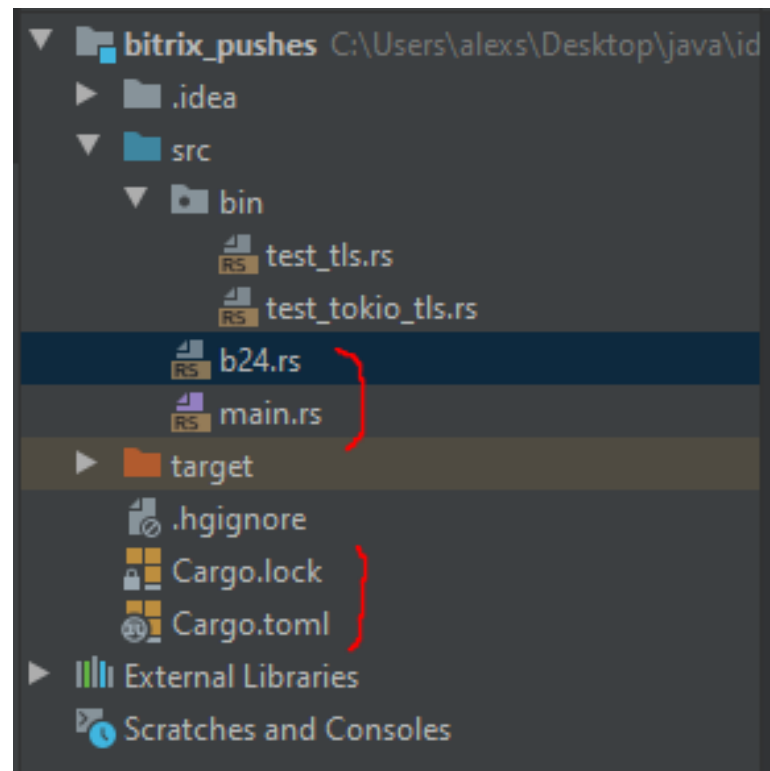
TCP – не кэшируются, ключи на диске – не кэшируются. Это еще сильнее снизит нагрузку.

Проект на Rust – обзор

- Стандартные зависимости в Cargo
- Сетевая асинхронная “монадическая” библиотека с корутинами – Tokio
- Немного танцев с бубнами с TLS, но исходники помогли быстро
- При росте нагрузки резко начались segfaults, но быстро полечились через увеличение **ulimit** ОС, видимо, наступили на ограничение стека на процесс, и наступала нирвана
- Никаких проблем, падений, анализов coredump
- Никаких утечек памяти, Rust использует стандартные: malloc или jemalloc или можно выбрать
- Никаких тюнингов GC и нагрузок от сборщика мусора 😊

Проект на Rust – обзор

```
[dependencies]
base64 = "0.10.1"
chrono = "0.4.9"
clap = "2.33.0"
futures = "0.1.29"
lazy_static = "1.4.0"
log = "0.4.8"
env_logger = "0.7.1"
regex = "1.3.1"
rusoto_sqs = "0.41.0"
rusoto_core = "0.41.0"
rustls = "0.16.0"
serde_json = "1.0.41"
signal-hook = "0.1.11"
tokio = "0.1.22"
tokio-core = "0.1.17"
tokio-rustls = "0.10.2"
webpki = "0.21.0"
webpki-roots = "0.18.0"
```



Проект на Rust – тесты

- Сразу пишутся тесты
- За счет ADT/PM, отсутствия NULLs, Enums – тесты больше занимаются делом
- В самом языке много вкусных функциональных библиотек с Zero-cost abstraction (итераторы, диапазоны, фильтры, мап-редьюсы)
- В Tokio тоже монадически-функциональный подход

```
1051 #[cfg(test)]
1052 mod tests {
1053     use super::*;
1054
1055     #[test]
1056     fn test_push_type() {
1057
1058         assert_eq!(PushType::Unknown.value(), 0);
1059         assert_eq!(PushType::APNSProd.value(), 1);
1060         assert_eq!(PushType::APNSProd.value(), 2);
1061         assert_eq!(PushType::Google.value(), 3);
1062         assert_eq!(PushType::AppleV2Sandbox.value(), 4);
1063         assert_eq!(PushType::AppleV2Prod.value(), 5);
1064
1065         assert_eq!(PushType::by_num(0), PushType::Unknown);
1066         assert_eq!(PushType::by_num(1), PushType::APNSProd);
1067         assert_eq!(PushType::by_num(2), PushType::APNSProd);
1068         assert_eq!(PushType::by_num(3), PushType::Google);
1069         assert_eq!(PushType::by_num(4), PushType::AppleV2Sandbox);
1070         assert_eq!(PushType::by_num(5), PushType::AppleV2Prod);
1071         assert_eq!(PushType::by_num(100), PushType::Unknown);
1072     }
1073 }
```

```
Finished test [unoptimized + debuginfo] target(s) in 2.44s
Running target/debug/deps/bitrix_pushes-4635fc3582b77339.exe

running 15 tests
test b24::amazon::tests::test_apci ... ok
test b24::amazon::tests::test_apci_response ... ok
test b24::amazon::tests::test_apiv2 ... ok
test b24::amazon::tests::test_apiv1 ... ok
test b24::amazon::tests::test_apci_p2 ... ok
test b24::amazon::tests::test_apci_p1 ... ok
test b24::amazon::tests::test_extract_raw_packets ... ok
test b24::amazon::tests::test_make_google_payload ... ok
test b24::amazon::tests::test_path2key_google ... ok
test b24::amazon::tests::test_push_type ... ok
test b24::amazon::tests::test_remove_spaces ... ok
test b24::amazon::tests::test_transform_raw_packets ... ok
test b24::amazon::tests::test_extract_transform ... ok
test b24::amazon::tests::test_transform_raw_packets2 ... ok
test b24::amazon::tests::test_check_tkey ... ok

test result: ok. 15 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out

Running target/debug/deps/test_tls-33786fbcca939adf.exe

running 0 tests

test result: ok. 0 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out

Running target/debug/deps/test_tokio_tls-b209aa9008c150fb.exe

running 0 tests

test result: ok. 0 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out
```


Проект на Rust – ADT

```
#[derive(Debug, PartialEq, Clone)]
enum AppleDeployKind {
    Dev,
    Prod
}

#[derive(Debug, PartialEq, Clone)]
enum AppleProtoVersion {
    One,
    Two
}

#[derive(Debug, PartialEq, Clone)]
struct ApplePushCertsInfo(AppleProtoVersion, AppleDeployKind);
```

Проект на Rust – Pattern Matching

```
fn tls_connect_info(&self) -> AppleTlsConnectInfo {  
    match *self {  
        ApplePushCertsInfo(  
            AppleProtoVersion::One, //old v.  
            AppleDeployKind::Dev  
        ) => AppleTlsConnectInfo {  
            url: String::from(s: "gateway.sandbox.push.apple.com"),  
            port: 2195,  
            folder: self.folder(),  
            certs_filename: String::from(s: "server_certificates_bundle_sandbox.pem")  
        },  
    },  
}
```

Проект на Rust – присваивание из PM

```
//we use tokio to async execute queries to AWS, Google, Apple
let mut tokio_core : Core = match Core::new() {
    Ok(c : Core) => {
        debug!("{}: Thread tokio event loop started.", std::thread::current().name().unwrap());
        c
    },
    Err(e : Error) => {
        // it is fatal not to start event loop
        error!(
            "{}: Error starting event loop: {}. Stop program.",
            std::thread::current().name().unwrap(),
            e
        );
        std::process::exit( code: 1 );
    }
};
```

Проект на Rust – “Option” monad

```
//sync - OK, as used only on thread launch
let queue_url : String = match client.get_queue_url( input: gqur).sync() {
    Ok(v : GetQueueUrlResult) => {
        v.queue_url.unwrap_or_else(|| {
            //stop process
            error!(
                "{}: Error unwrap SQS queue URL. Exit program.",
                std::thread::current().name().unwrap()
            );
            std::process::exit( code: 1);
        })
    },
}
```

```
let f : MapErr<AndThen<..., fn(...)> = socket
    .and_then(
        move |ts : TcpStream| {

            //to fasten writes
            let _res = ts.set_nodelay(true).unwrap();

            let d : String = tci.url.clone();
            let domain : DNSNameRef = webpki::DNSNameRef::try_from_ascii_str( dns_name: d.as_str()).unwrap();

            trace!(
                "{}: TCP connect established, begin TLS (Apple): {:?}",
                std::thread::current().name().unwrap(),
                d
            );

            tc.connect(domain, stream: ts)

        }
    ).and_then(
        |ts : TlsStream<TcpStream>| {

            trace!(
                "{}: TLS connect established, begin write data (Apple)",
                std::thread::current().name().unwrap()
            );

            tokio::io::write_all( a: ts, buf: bytes)

        }
    ).and_then(
        |_, buf : Vec<u8>| {
```

```
    .map_err(move |e : Error| {
        error!(
            "{}: Error send a push (Apple): {:?} {:?} cur_errs_cnt={:?}",
            std::thread::current().name().unwrap(),
            e,
            tkey_cl,
            crate::ST_APPLE_PUSH_ASYNC_SEND_ERR.load(order: Ordering::SeqCst)
        );
        crate::ST_APPLE_PUSH_ASYNC_SEND_ERR.fetch_add(val: 1, order: Ordering::SeqCst);
    })
);

//ask to send async
tokio_handle.spawn(f);
```

Zero-cost – функциональщина

```
//Remove spaces from a key content
fn remove_spaces(s: &String) -> String {

    s.chars().filter(|c : &char | {

        if *c == ' ' || *c == '\r' || *c == '\n' || *c == '\t' {
            false
        } else {
            true
        }
    }).collect()

}
```


ООП? Traits!

- Развитого ООП, как в Java/C# – нет и не планируется. А зачем?
- Структурки и методы, навешанные на них
- Но за счет строгой типизации, ADT/PM, аффинных типов и сильных гарантий компилятора – этого хватает

```
#[derive(Debug, PartialEq)]
```

```
struct ApplePushInfoV1 {
```

```
    command: u8,
```

```
    token_length: u16,
```

```
    device_token: Vec<u8>,
```

```
    payload_length: u16,
```

```
    payload: Vec<u8>
```

```
}
```

```
impl ApplePushInfoV1 {
```

```
    fn default() -> ApplePushInfoV1 {
```

```
        ApplePushInfoV1 {
```

```
            command: 0,
```

```
            token_length: 0,
```

```
            device_token: vec![],
```

```
            payload_length: 0,
```

```
            payload: vec![]
```

```
        }
```

```
    }
```

```
    fn parse(b: Vec<u8>) -> Result<ApplePushInfoV1, String> {
```

А что, если взять...

- Node.js
- Python – корутины
- Golang – горутины
- Java/Netty
- Erlang
- Haskell
- C++



Подводные камни

- Rust далеко не всем «по зубам» – крутая кривая входа. Большинство разработчиков склоняются к Golang.
- В Rust легче перейти с имеющимся развитым функциональным бэкграундом (для haskell), чем из прикладного скриптинга.
- На Golang описанную задачу можно решить быстрее и кода получается меньше, а по скорости различие не на критических нагрузках – не такое заметное. Но различие – есть, особенно по CPU.
- В Golang стандартная библиотека решений кажется более развитой, хотя Cargo тоже активно и постоянно развивается.
- C++ или Rust? Что там с написанием драйверов под Linux на Rust? Что с Android.
<https://habr.com/ru/company/selectel/blog/552142/>

В результате у нас больший приоритет отдается проектам на Golang, чем на Rust.

Итоги

- Rust только кажется сложным. За сложностью скрывается строгость, простота и элегантность
- Язык привносит несколько очень мощных и непривычных концепций, направленных на надежное программирование, в т.ч. системное
- Компилятор дает очень высокие гарантии по работе с памятью и многие другие (переполнения, доступ за пределы массива ...) и его нужно «преодолевать»
- На языке легко писать, причем быстро и сразу без ошибок
- Алгебраические типы данных, строгий pattern-matching и отсутствие NULLs хорошо структурируют код и сильно ускоряют разработку
- В Cargo «из коробки» много готовых модулей, бери и используй
- Скорость решения сравнима с C, потребление ОЗУ без сборщика мусора минимально

Спасибо за
внимание!
Вопросы?

Александр Сербул

 @AlexSerbul

 Alexandr Serbul

serbul@1c-bitrix.ru

